Argonne NATIONAL LABORATORY

ANL/MCS-TM-312

2009 Fault Tolerance for Extreme-Scale Computing Workshop

Albuquerque, NM – March 19-20, 2009

Mathematics and Computer Science Division
Technical Memorandum ANL/MCS-TM-312

ANL/MCS-TM-312

2009 Fault Tolerance for Extreme-Scale Computing Workshop

Albuquerque, NM – March 19-20, 2009

by
D.S. Katz,[1,2] J. Daly,[3] N. DeBardeleben,[4] M. Elnozahy,[5] B. Kramer,[6] S. Lathrop,[7,8]
N. Nystrom,[9] K. Milfeld,[10] S. Sanielevici,[9] S. Scott,[11] and L. Votta[12]
[1] Computation Institute, University of Chicago & Argonne National Laboratory
[2] Center for Computation & Technology, Louisiana State University
[3] Center for Exceptional Computing
[4] Los Alamos National Laboratory
[5] IBM
[6] NCSA, University of Illinois
[7] Argonne National Laboratory
[8] Shodor Foundation
[9] Pittsburgh Supercomputer Center
[10] Texas Advanced Computing Center
[11] Oak Ridge National Laboratory
[12] Brincos Inc.

December 2009

This work was supported in part by the U.S. Dept. of Energy under
Contract DE-AC02-06CH11357.

Table of Contents

2009 Fault Tolerance for Extreme-Scale Computing Workshop

Albuquerque, NM - March 19-20, 2009

Daniel S. Katz,[1,2] John Daly,[3] Nathan DeBardeleben,[4] Mootaz Elnozahy,[5] Bill Kramer,[6] Scott Lathrop,[7,8] Nick Nystrom,[9] Kent Milfeld,[10] Sergiu Sanielevici,[9] Stephen Scott,[11] Lawrence Votta[12]

Abstract

This is a report on the third in a series of petascale workshops co-sponsored by Blue Waters and TeraGrid to address challenges and opportunities for making effective use of emerging extreme-scale computing. This workshop was held to discuss fault tolerance on large systems for running large, possibly long-running applications. The main point of the workshop was to have systems people, middleware people (including fault-tolerance experts), and applications people talk about the issues and figure out what needs to be done, mostly at the middleware and application levels, to run such applications on the emerging petascale systems, without having faults cause large numbers of application failures. The workshop found that there is considerable interest in fault tolerance, resilience, and reliability of high-performance computing (HPC) systems in general, at all levels of HPC.

The only way to recover from faults is through the use of some redundancy, either in space or in time. Redundancy in time, in the form of writing checkpoints to disk and restarting at the most recent checkpoint after a fault that cause an application to crash/halt, is the most common tool used in applications today, but there are questions about how long this can continue to be a good solution as systems and memories grow faster than I/O bandwidth to disk. There is interest in both modifications to this, such as checkpoints to memory, partial checkpoints, and message logging, and alternative ideas, such as in-memory recovery using residues.

We believe that systematic exploration of these ideas holds the most promise for the scientific applications community. Fault tolerance has been an issue of discussion in the HPC community for at least the past 10 years; but much like other issues, the community has managed to put off addressing it during this period. There is a growing recognition that as systems continue to grow to petascale and beyond, the field is approaching the point where we don't have any choice but to address this through R&D efforts.

---

[1] Computation Institute, University of Chicago & Argonne National Laboratory
[2] Center for Computation & Technology, Louisiana State University
[3] Center for Exceptional Computing
[4] Los Alamos National Laboratory
[5] IBM
[6] NCSA, University of Illinois
[7] Argonne National Laboratory
[8] Shodor Foundation
[9] Pittsburgh Supercomputer Center
[10] Texas Advanced Computing Center
[11] Oak Ridge National Laboratory
[12] Brincos Inc.

Introduction

This is the third in a series of petascale workshops co-sponsored by Blue Waters and TeraGrid to address challenges and opportunities for making effective use of emerging extreme-scale computing.

The purpose of this workshop was to discuss fault tolerance on large systems for running large, possibly long-running applications. The main point of the workshop was to have systems people, middleware people (including fault-tolerance experts), and applications people talk about the issues and figure out what needs to be done, mostly at the middleware and application levels, to run such applications on the emerging petascale systems, without having faults cause large numbers of application failures.

One way of looking at the topic, and the potential discussion, is through the following questions (as suggested by Bill Gropp):

- How serious is the problem? This must *not* be the usual simplistic analysis based on the failure rates of commodity server nodes; it needs to look more closely at what happens in large systems where the designers take into account the scale of the system.
- How does one know whether there is a fault (does one need to do more than trust the system)?
- What automatic methods (e.g., system-level checkpoint/restart) are there, and what are their pros and cons, particularly with respect to performance impact and portability?
- What tools are available to manage user-level checkpoint/restarts?
- What algorithmic approaches are there to detect and repair faults as alternatives to checkpoint/restart?

All information from the workshop, including presentation slides, is available at http://www.teragridforum.org/mediawiki/index.php?title=2009_Fault_Tolerance_Workshop.

Organizers

Daniel S. Katz, LSU/U. Chicago (Blue Waters/TeraGrid)
Scott Lathrop, ANL/NCSA (Blue Waters/TeraGrid)
Bob Wilhelmson, NCSA (Blue Waters)
Nick Nystrom, PSC (TeraGrid)
Amit Majumdar, SDSC (TeraGrid)
Sergiu Sanielevici, PSC (TeraGrid)
Patrick Bridges, UNM
Carolyn Peters, ANL (TeraGrid)

Workshop Talks

Fault Tolerance 101, Zbigniew Kalbarczyk (U. Illinois)

> The talk provided an introduction to the basic concepts in designing dependable systems. We begin with the basic terms including reliability, availability, mean time to failure, and hardware and software fault models. Then the presentation discusses the importance of low-latency detection to limit uncontrolled error propagation and enable rapid recovery. The basic types of redundancy---spatial, information and time---are illustrated by example fault-tolerance techniques that use a given type of redundancy. The techniques discussed include triple modular redundancy, replication, coding, and checkpointing. We conclude by emphasizing the need for a system view when designing fault-tolerant systems; that is, to build a robust fault-tolerant system, one must consider both system hardware (processor, memory) and software (OS and applications) and understand how the two interact.

Faults and Fault-Tolerance on the Argonne BG/P System, Rinku Gupta (ANL)

> The Argonne Leadership Computing Facility (ALCF) houses the 550+ Tflops IBM Blue Gene/P ``Intrepid'" system. This talk presented some experiences with this system, with a focus on observed faults, fault management, and fault handling. The talk also discussed the fault-related challenges being exposed by emerging multi-petascale machines and research being done to handle those challenges. One such research effort is the Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) initiative. In contrast to current methods where all software handles faults independently, CIFTS provides a foundation to enable systems to adapt to faults in a holistic manner by providing a coordinated infrastructure that allows all system software to share fault-related information. This talk also presented the research done by the CIFTS group for software used on these high-end computing machines.

Essential Feedback Loops, Jon Stearley (SNL)

> Fault tolerance is not a systems problem, nor an apps problem---it is *our* problem. Real solutions will require workgroup and software feedback loops among systems and apps. Establishing standard measurements of HPC fault tolerance is a critical enabling step: our terminology overlaps, but our definitions are inconsistent. Open sharing of relevant raw data is also key, such as via http://cfdr.usenix.org. This talk reviewed these topics, as well as described failure rates and modes on Red Storm at Sandia National Laboratories.

Experiences with Kraken, Patricia Kovatch (NICS)

> The Malthusian catastrophe is upon us! Application problem size and machine complexity are growing geometrically, while mitigation techniques (MTBF of individual components, etc.) are improving linearly. Thomas Malthus, an English political economist who lived from 1766 to 1834, observed that populations tend to grow geometrically while food supply grows only linearly. He therefore predicted that populations will be limited by starvation. As this hasn't happened yet, we explored the parallels and lessons from the agriculture and applied them to HPC.
>
> Some of the techniques that agriculture has adopted to feed the growing population include taking advantage of economies of scale (concentrating resources and making large infrastructure investments), developing wider markets and better distribution networks (to survive a drought in one area), and designing more efficient technologies (like tractors). People also developed techniques like canning to store food in case of hard times. The HPC community has similarly evolved by operating fewer and more efficient centers, opening up networked resources (TeraGrid, DEISA, OSG), and taking advantage of better machine design (large industrial-strength fans per cabinet instead of many PC-quality fans, reducing the overall number of parts). Users also checkpoint their applications in case of failure.

Growing Pains of Petascale Computing: Integrating Hardware, Software and Middleware for Successful Capacity and Throughput, Kent Milfeld (TACC)

This talk discussed how fault tolerance and errors are a growing problem at TACC, as systems (and investments) grow larger and as work done on the systems becomes more mission critical and timely. The talk also included details of failure rates of various components (CPUs, memory, switch chips, flash, and disk) on TACC's Ranger.

Rollback-Recovery in the Petascale Era, Mootaz Elnozahy (IBM)

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted because of failures and recoveries. Typical MTBF is from 8 hours to 15 days. As systems increase in size to field petascale computing capability and beyond, the MTBF will go lower, and more capacity will be lost. The current state of the art relies on checkpoint-rollback in the context of a flat message-passing model. This talk contended that this combination is inherently nonscalable and we must take a serious look at alternatives. The speaker claimed that the lack of scalability is fundamental in this model, because of the lack of any sort of failure containment. And while good engineering is necessary and will be useful to reduce the waste, they believe that a fundamentally new model is needed for large-scale systems going forward. He noted that this starts with a programming model that has a structure that enables scalability and failure containment.

The speaker also analyzed the new trends in technology and showed that power management, the recent trend toward heterogeneous computing, and the expected increase in system size all will interact negatively with resilience at the system level. They believe that failures will no longer be "nice" fail-stop failures or crash failures. Instead, we will see an increase in soft errors that may escape the first level of detection and correction at the hardware. We also will see new programming models based on hybrid software and hardware systems, and we will see aggressive use of power management at all levels. The talk contended that providing reliability in these contexts remains an open research problem.

Large Systems Experience, Bill Kramer (NCSA)

Systems that support highly parallel applications are more tightly constrained regarding failure identification, diagnosis, and recovery than are systems of similar scale that support lower levels of parallelism. Petascale systems, with massive numbers of components and many layers of software, need new analysis and approaches in order to provide reliable service, including better understanding of failure causes, better information channels to the system and applications about failures and better handling of failures. This talk began by reviewing causes of systemwide and application-level failure for current HPC systems. Data shows software is more likely to cause systemwide outages than is hardware, for example. Data also shows certain SW and HW components exhibit higher error rates than do other components regardless of system or vendor. Then the discussion moved to possible mitigations and/or solutions for the Petascale systems coming on the scene, such as Blue Waters.

The Role of Storage in Exascale Fault Tolerance, Garth Gibson (CMU)

The SciDAC Petascale Data Storage Institute (PDSI, http://www.pdsi-scidac.org/) is charted to address the challenges of petascale computing for scientific discovery on information storage capacity, performance, concurrency, reliability, availability, and manageability. Checkpoint capture bandwidth, an integral component in the dominant fault-tolerance strategy, is the most immediately challenging performance requirement for extreme-scale applications. The causes of this are (1) a balanced system design that calls for storage bandwidth to grow at the rate that memory size and compute speed grow, while the individual bandwidth of disks grows more slowly than the individual speed of processor chips or the size of memory chips, and (2) the increasing numbers of components in extreme-scale computers, which are expected to yield decreasing time until interrupt, threatening to lower the effective fraction of compute cycles available to applications. This talk reviewed these challenges, plus the internal fault-tolerance challenges in scalable storage aggravated by the growing capacity of each disk. Moreover, a popular checkpoint representation is the single file containing all checkpoint data, which can lead to concurrent writing from thousands of nodes, and, for some applications, a small strided writing pattern generally synchronized in the same region of the single file at the same time. This write pattern is challenging for some file systems, or can cost relatively high-capacity overhead where mirroring is used as a solution. New "log-structured" representations are being developed by a variety of projects, in which data is stored not by address in the file but by order of writing. This talk also presented a new approach to write-optimized checkpoint capture, the Parallel Log-structured File System (PLFS), a FUSE-based interposition file system that represents a checkpoint as a hidden directory of logs written by each writing thread, led by a team at Los Alamos National Laboratory.

I/O Fault Diagnosis in Software Storage Systems, Eric Schrock (Sun)

Be wary of your storage system, and learn to love checksums. This talk discussed some design principles for systems capable of identifying and managing hardware faults, with a focus on software storage systems. General topics included structured event logs and diagnosis, physical identification of failed components, proactive response, and unified presentation of all hardware faults on a system. Additional storage-specific topics included diagnosis throughout the I/O stack, block-independent checksums, and integrated volume managers to enable automated repair of data.

stdchk: A Checkpoint Storage System for HPC Applications, Sudharshan Vazhkudai (ORNL)

Stdchk is a checkpoint storage device, built from memory resources, that acts as an intermediary to the central parallel file system. The working hypothesis is that such a system could be built from a percentage of a user's own allocation or even residual memory resources. The system comprises a dedicated manager that aggregates memory resources from processors (benefactors) and makes it available as a collective space for checkpointing clients, using a standard POSIX file system interface using FUSE. Such a system has the potential to alleviate the I/O bandwidth bottleneck for bursty checkpoint I/O operations by aggregating memory and interprocessor bandwidth. The aggregate memory buffer needs to create room for incoming snapshot data. Thus the data in aggregate memory is drained asynchronously to stable storage.

Benefactor PEs can come and go depending on volatility of the processor. To accommodate this transient behavior, benefactors register with the manager using a soft-state registration protocol. The manager keeps track of the memory contributions from the benefactors and presents a unified storage space to client PEs. The manager maintains metadata regarding individual benefactor contributions, each benefactor's status, and potentially some history about the benefactor. When a client contacts the manager, the file to write is divided by the system into equal-sized chunks. The manager computes a striping plan, determining a set of benefactors to send the chunks to, and a benefactor mapping. Once clients obtain a striping ``map,'' they interact with the benefactors directly to send the chunks to benefactors. Experiments on a 10 Gb/s cluster interconnect show a peak write performance of around 700 MB/s.

Stdchk supports similarity detection (compare-by-hash) between successive checkpoint images in order to minimize the amount of data written during each timestep. A simple, yet elegant, strategy is to compute a hash of the chunks and to store them as metadata at the manager at each timestep, t. At the (t+1)th timestep, the hashes for chunks of the checkpoint image are compared against the stored metadata to detect similarity. If the chunk hashes are similar, then the new chunk in question need not be written, but only a reference to the old chunk needs to be retrieved. The hash comparison is a metadata operation that can be performed between the client and the manager and does not involve the chunks stored at the benefactors. Consequently, the snapshot data from timestep, t, need not be maintained in the aggregate memory during timestep t+1. Experiments suggest a reduction in size of up to 24% for checkpoints written by using BLCR at 15-minute intervals.

Future plans for stdchk include features such as automatic pruning of checkpoint images based on application hints to reduce the amount of checkpoint data from a single run and on-the-fly instantiation of the storage system at job startup.

System-Level Checkpoint/Restart with BLCR, Paul Hargrove (LBNL)

Berkeley Lab Checkpoint/Restart (BLCR for short) is an open source software implementation of system-level checkpoint/restart for Linux clusters, which when combined with a supported MPI implementation is able to provide preemptive checkpointing of unmodified MPI applications. The preemptive nature of BLCR allows for use of checkpoint/restart and migration for scheduling purposes, in addition to the classic fault-tolerance uses. BLCR is currently available for the 32- and 64-bit versions of the x86 and PPC architectures and the 32-bit ARM embedded processor. The 2.6.x Linux kernels from all major distributions are supported, including CNL on the Cray XT series, while support for ANL's ZeptoOS for the IBM BG/P is in development. This talk provided a brief introduction to the goals and approach taken by the BLCR team and described the current status and future work, including live-migration and multiple approaches to reduce the I/O requirements: compression, MMU-based "incremental" checkpoints, and checksum-based "differential" checkpoints.

Scalable Fault-Tolerance Schemes Using Adaptive Runtime Support, Celso Mendes (U. Illinois) and Laxmikant V. Kale (Univ. of Illinois, Urbana-Champaign)

HPC systems for computational xcience and engineering have almost reached the threshold where some form of fault tolerance becomes mandatory. Although system-level checkpoint-restart keeps things simple for the application developer, it leads to high overhead. Meanwhile, application-level schemes are effort-intensive for the programmer. Schemes based on smart runtime systems appear to be at the right level for addressing fault tolerance. Our work, based on object-level virtualization and implemented by the Charm++ runtime system, provides such a scheme.

Charm++ offers a series of techniques that can help tolerate faults in large parallel systems. These techniques include distributed checkpoints, message-logging with parallel recovery, and proactive object migration. When combined with the measurement-based load balancing facilities available in Charm++, one can both tolerate faults and continue execution on remaining resources with optimal efficiency. These techniques can also be applied to MPI applications running under AMPI, an MPI implementation based on Charm++.

Handling Faults in a Global Address Space Programming Model, Sriram Krishnamoorthy (PNL)

A common feature of programming models targeting high productivity is a global address space that allows noncollective access to global data. While MPI has often been the focus of fault-tolerance research, handling faults in the context of such models is becoming increasingly important. This talk presented the authors' investigation into handling faults in global address space programming models.

They have developed a version of the Aggregate Remote Memory Copy Interface (ARMCI) one-sided communication library that is capable of suspending and resuming applications, by exploiting efficient support for the Infiniband network in the Xen virtual machine environment. They are investigating a taskpools-based approach to fault resilience, where failures can be survived and the performance penalty incurred due to failure is proportional only to the number of faults. Such a fault-resilience mechanism requires support from the communication subsystem to identify failures, selectively restart processes, and provide feedback to higher layers on the state of the global address space. They are currently investigating ways of providing such support in ARMCI and the Global Arrays library.

Implications of System Errors in the Context of Numerical Accuracy, Patty Hough (SNL)

Establishing credibility is essential for predictive simulations used to support high-consequence decisions. Verification and validation play key roles in establishing that credibility, and one aspect of code verification is to determine whether numerical algorithms are implemented and functioning properly. To date, the authors have not explicitly addressed soft errors (e.g., bit flips) as part of this assessment, but they expect these errors to become more prevalent as architectural features continue to shrink and operate at low voltages.

To get a sense of the extent to which a bad floating-point operation can affect numerical algorithms, the authors conducted a simple experiment using GMRES, a Krylov method that serves as the workhorse for many of our simulation codes. A single bad floating-point operation in the subspace construction led to a solution that was incorrect by several orders of magnitude. A bad operation in computing the orthogonal basis led to nonconvergence. Such results indicated that the impact of soft errors warrant further study, including assessing the risk of these errors occurring during computation, determining how they propagate through the entire simulation, and exploring computational models such as transactional computing coupled with guaranteed data regions.

The Scalable Checkpoint / Restart (SCR) Library: Approaching File I/O Bandwidth of 1 TB/s, Adam Moody (LLNL)

The Scalable Checkpoint / Restart (SCR) library provides an interface to codes that many use to write out and read in preprocess, application-level checkpoint files in a scalable fashion. In the current implementation, checkpoint files are cached in local storage (hard disk or RAM disk) on the compute nodes. This technique provides scalable aggregate bandwidth and uses storage resources that are fully dedicated to the job. This approach addresses the two common drawbacks of checkpointing a large-scale application to a shared parallel file system, namely, limited bandwidth and file system contention. In fact, on current platforms, SCR scales linearly with the number of compute nodes. It has been benchmarked as high as 720 GB/s on 1,094 nodes of Atlas at LLNL, which is nearly two orders of magnitude faster than the parallel file system. The library applies a redundancy scheme to the checkpoint data, and it supports the use of spare nodes such that it is capable of rebuilding the cached checkpoint set and restarting the job even after a node fails. In particular, this restart can complete without the need to write out and read in the checkpoint file set via the parallel file system. By using this library, large-scale productions runs of pf3d have reduced their checkpoint times from 1,200 seconds to 15 seconds, reduced their percentage of runtime spent checkpointing from 25% to 5%, reduced their checkpoint I/O to the parallel file system by a factor of 7, and increased their mean-time-before-failure from 1.5 hours to tens of hours.

The SCR library is written in C, and it currently provides only a C interface. It is well tested on Linux clusters with RAM disc and local hard drives, and it has been used in production at LLNL since late 2007 on Linux AMD Opteron clusters with Infiniband. The current implementation is closely tied to SLURM. The library is designed to be portable on other platforms and other resource managers. It is an open source project licensed under BSD hosted at
http://sourceforge.net/projects/scalablecr.

Sustained Exascale: The Challenge, George Bosilca (U. Tennessee)

Even making generous assumptions about the reliability of a single processor, it is clear that as the processor count in high-end clusters grows into the tens of thousands, the mean time to failure (MTTF) will drop from hundreds of days to a few hours or less. Although today's architectures are robust enough to incur process failures without suffering complete system failure, at this scale and failure rate the only technique available to application developers for providing fault tolerance, within the current parallel programming model of checkpoint/restart, has performance and conceptual limitations that make it inadequate to the future needs of the communities that will use these systems.

The MPI standard, as defined by the MPI forum in 1994-1997, has undoubtedly become the de facto standard for parallel applications. Though the MPI standard does not provide any help to recover from failures, it defines several error codes returned by MPI functions to warn the application that may be used to take some corrective action. Several projects have aimed at easing the development of fault-tolerant MPI applications. Although work in fault tolerance has been dominated in recent years by systems-oriented approaches that are transparent to the application and relatively easy to use, the authors believe that, in the next generation of high-end computing environments, successful approaches to fault tolerance must leverage intimate knowledge of the parallel environment, the application, and its dominant algorithm in order to achieve the efficiencies required for faster (or even adequate) recovery times and memory requirements. This talk presented the latest advances in uncoordinated checkpoint/restart as well as user-level recovery algorithms based on the research and prototypes from the University of Tennessee.

Between Application and System: Fault Tolerance Mechanisms for the Cactus Software Framework, Erik Schnetter (LSU)

After giving a brief overview over the structure and main features of Cactus, the authors presented some numbers for the way in which Cactus is today typically used for production HPC simulations in astrophysics. The current way to address fault tolerance in Cactus is by checkpointing and recovery, which is sufficient for current levels of reliability and which is needed to stay within the time slot limits imposed by queuing system policies. Since checkpointing/recovery is provided by the framework and neither the operating system nor the end user, it can be both reliable and efficient. Cactus's modular nature means that virtually all Cactus users are also developers; hence, tools and infrastructure for debugging, profiling, or fault tolerance need to involve the end user and not only experts.

The speaker said that he assumed that fault tolerance will become an issue only as the scalability of Cactus applications is significantly increased, and scalability and fault tolerance are therefore two sides of the same issue. Cactus users request want mundane features, such as not losing a queue slot when a simulation aborts. It is also necessary to improve system-level error messages, which are often very low level and lack detail. From a user perspective, there is a smooth transition between debugging, resource exhaustion, and fault tolerance. Unfortunately, error messages are often insufficient to find the cause of an error.

The speaker raised the question: Should there be a common measure ("benchmark") for application fault tolerance? If so, what quantities would be measured, and how could they be measured on today's systems?

Fault Tolerance Support for HPC Tools and Applications: Scalability and Sustainability, Tony Drummond (LBNL)

This talk looked at three case studies---the DOE ACTS collection and two applications in climate modeling: high-resolution atmospheric modeling and coupled climate simulations---all in the context of sustainable software development. The Advanced CompuTational Software Collection (ACTS) makes reliable and efficient software tools more widely used and more effective in solving the nation's engineering and scientific problems. Part of the software sustainability process is automated testing. For the high-resolution atmospheric modeling application, there is currently no checkpointing, and no immediate plans to add it. For the coupled climate simulations, checkpointing is possible but difficult. The common needs from these three case studies are lightweight FT interfaces to recover from identifiable, reproducible, detectable or predictable errors that happen at 1K+ cores, including support for scalable archival for applications at the exascale and beyond, and portable tools (available everywhere, specially emerging systems). Questions remain on how complex applications can define FT and recoverability and on issues related to having/using spare cores and dealing with queueing/scheduling systems.

Fault Tolerance in CREATE Phase 1, Lawrence Votta (Brincos Inc.)

Computational Research and Engineering Acquisition Tools and Environments (CREATE) is a 12-year program to develop and deploy three computational engineering toolsets for acquisition engineers: Air Vehicle, Ship; and RF Antennas. The basic value proposition of CREATE is to replace the general weapons acquisition workflow of physical design-build-test with computational design-build-test iterations validated with a final physical test. This will better position the DoD acquisition process to leverage modern design processes (driven by affordable petascale computing) and greatly reduce cost, design intervals and late identification of design limitations and shortfalls.

CREATE will be developed in two phases: a legacy phase and a new code phase. The legacy phase started in 2008 and collects the current computational models for each of the three toolsets above. Initial work on these three sets will be to provide (1) state-of-practice development and design environments and (2) collaboration tools and environments. The majority of this work will transition in 2013 to one specializing in development of production-quality, physics-based CSE tools using the knowledge from the technology of current tools and how they are used. Opportunities for software tool reuse will be identified and incorporated into Phase 2.

One important area of learning in Phase 1 of CREATE will be to understand the structure of the computations on modern computing platforms, so that computational integrity can be maintained as elements of the computer hardware and operating system experience transient and permanent failures. The authors envisioned using the discrete time simulation of the computations to provide checkpoints from which computations can be both examined and restarted if needed. Checkpoints are a classic fault-tolerance technique and have been studied in both the computer architecture area (see Modeling coordinated checkpointing for large-scale supercomputers, Wang et al, Proceedings of International Conference on Dependable Systems and Networks, 2005, pages 812-821) and computational science area (see Daly, Proceedings of the First International Workshop on Software Engineering for High Performance Computing System Applications, 2004 pages 70-74).

Although these methods can lead to an application with fault tolerance, there are many challenges centered on (1) the productivity of the application developer and user community and (2) assumptions about how, when, and where the computer hardware and operating system detect, isolate, contain, and recover from transient and permanent failures. This talk highlighted some of these challenges and potential opportunities for solution in the hardware, system, middleware, and application software.

Application Resilience for Truculent Systems, John Daly (Center for Exceptional Computing)

High-performance computing is confronted by at least three progressively consequential challenges to traditional methods of fault tolerance. The most obvious is the impact of increasing component counts and densities on the frequency and severity of permanent, intermittent, and transient system faults. As a consequence of increasing failure rates, the overhead associated with redundancy-based fault-tolerant schemes is increasing the cost---in terms of hardware, software, power, and dollars---required to provide fault tolerance. Perhaps more subtly, a proliferation of dependent and silent failure modes reduces the ability of monitoring to accurately measure system state and infer application progress, thus posing unique challenges for application performability and correctness.

The speaker argued that the traditional paradigm of system fault-tolerance needs to be supplanted by an application-centric paradigm of resilience in order to facilitate correct solutions in a timely and efficient manner in the presence of increasingly frequent and enigmatic system degradations and failures. While traditional fault tolerance segregates the system and the application and addresses their failure modes independently, resilience is an integrated approach in which the system and the application confront failure modes collaboratively. Fault tolerance is focused on failure recovery, but resilience uses a synthesis of monitoring, analysis, and response to enable the application to circumvent system failures.

Insights into Fault Tolerance from FLASH Production Runs on Top 3 Supercomputing Platforms, Don Lamb (U. Chicago)

The speaker reported insights into fault tolerance gained from production runs using FLASH, a highly capable, fully modular, and extensible community code, on Top 3 supercomputing platforms. The three platforms are the BG/L at LLNL, the Franklin Cray XT4 at NERSC at LBNL, and the Intrepid BG/P at ALCF at ANL. The FLASH production runs simulated three different problems: (1) homogeneous, isotropic, weakly compressible, driven turbulence; (2) buoyancy-driven turbulent nuclear combustion; and (3) Type Ia (nuclear powered) supernovae. The talk included movies of these simulations. The speaker described experiences running on these platforms. FLASH handles astronomically large ranges of values of physical quantities, uses high-level libraries (e.g., HDF5), and almost always operates at the upper limit of available memory (since it is used to simulate astrophysical problems). Consequently, FLASH has historically walked into almost every hardware and software limitation in these platforms. FLASH uses rollback and recovery through application checkpointing, but this can be expensive if failure occurs just before checkpointing. Fault-tolerant-backplane software that could keep the application informed about the state of the machine and use this knowledge to write a checkpoint before imminent failure would solve this problem.

Analysis of Cluster Failures on Blue Gene Supercomputing Systems, Christopher Carothers (RPI) & Thomas Hacker (Purdue)

Large supercomputers are built today with thousands of commodity components and suffer from poor reliability because of frequent component failures. The characteristics of failure observed on large-scale systems differ from smaller scale systems studied in the past. One striking difference is that system events are clustered temporally and spatially, which complicates failure analysis and application design. Developing a clear understanding of failures for large-scale systems is a critical step in building more reliable systems and applications that can better tolerate and recover from failures. The speakers analyzed the event logs of two large IBM Blue Gene systems, statistically characterized system failures, presented a model for predicting the probability of node failure, and assessed the effects of differing rates of failure on job failures for large-scale systems. The work presented should be useful for developers and designers seeking to deploy efficient and reliable petascale systems.

Overall Discussion

What are the fault- tolerance interests among the community?

The workshop began with introductions by all of the participants and a short description of their interests in attending the workshop. These interests include the following:

- Quantitative assessment of where the pain in addressing fault tolerance is today and will be in the future
- From applications perspectives, the requirements for scaling to petascale that are not yet being met
- Addressing resilience and looking toward how to mitigate failures in emerging applications to keep operating under failure conditions
- Advice for apps already running on 100K cores: how to handle faults
- Software solutions for addressing fault tolerance, at all levels: system software, middleware, applications
- Finding scalable solutions for checkpointing
- How to scale I/O systems and manage faults that come with both more components and more data
- Fault tolerance as the driving factor for future storage
- Education of today's students to be ready for these environments
- Use of data mining to help understand failure statistics
- Applying research findings to production systems
- Implications of failure on numerical reliability and libraries
- Component failure versus system failure

What is fault tolerance?

A number of terms and definitions were introduced during the workshop. A few of the commonly used terms are included here for reference.

Fault tolerance and its dependability measures include the following:

- Reliability – a measure of the continuous delivery of service; expected value is MTTF (mean time to failure)
- Maintainability – a measure of the service interruption; expected value is MTTR – mean time to repair
- Mean time between failure – MTBF=MTTR+MTTF
- Availability – a measure of the system delivery with respect to the alternation of the delivery and interruptions; expected value is MTTF/(MTTF+MTTR)
- Safety – a measure of the time to catastrophic failure expected value – MTTCF – mean time to catastrophic failure

Fault tolerance addresses detection of faults and recovery from faults. Fault classes can be categorized based on the following:

- Temporal persistence
    - Permanent faults, the presence of which are continuous and stable
    - Intermittent faults – the presence of which are only occasional, due to unstable hardware or varying hardware and software states
    - Transient faults – resulting from temporary environmental conditions
- Origin
    - Physical faults – stemming from physical phenomena internal to the system such as threshold change, shorts, and opens, or from external changes such as environmental electromagnetism, vibration, and radiation
    - Human-made faults – resulting from design faults, (introduced during system design, modification, or establishment of operating procedures) or interaction faults (such as violations of operating or maintenance procedures)

Dependable computing:

- The original definition of dependability stresses the need for justification of trust—states that dependability is the ability to deliver service that can justifiably be trusted.
- The alternative definition (that provides the criterion for deciding if the service is dependable) states that the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

Dependability (and security) attributes include:

- Availability – readiness for correct service
- Reliability – continuity of correct service
- Safety – absence of catastrophic consequences
- Confidentiality – absence of unauthorized disclosure of data
- Integrity – absence of improper system alternation
- Maintainability – ability to undergo modifications and repairs

The means for dependability include fault prevention, fault tolerance, fault removal, and fault forecasting. The threats include faults, errors, and failures.

Fault latency is the time between when a fault occurs and an error, when the fault becomes active. Error latency is the time between an error, when a fault becomes active, and when the error/failure is detected, such as a parity error. This is the period of error propagation, where errors can potentially infect other hardware or software components in the system. It is important to note that the duration of this propagation period cannot be measured on real systems.

Resilience is an application centric computing paradigm for keeping applications running to a correct solution in a timely and efficient manner in the presence of system degradations and failures. How is resilience different from fault tolerance? Fault-tolerance is segregated in that the system or application must keep itself running in spite of component failure; resilience is integrated in that the system works with applications to keep them running in spite of failure.

What are trends the community is experiencing?

Hardware complexity: The complexity of computing systems growing to 100K processors and above will cause the MTBF to rise. The types of errors that cause faults include boot failures and disk failures, with disk failures being the component causing the most interrupts on many systems. Boot failures can arise as the system partition is booted with a new copy of the operating system with every job or as a result of network and parallel file system issues that are not 100% resolved. Many say that the filesystem is becoming the component most likely to fail. Companies like Google are addressing high-quality and reliability standards for huge filesystems to serve their global audience; even though HPC problems are different and use filesystems differently, the HPC community should pay more attention to these examples.

Software complexity: Scalability requirements drive programming models and reliability measures. Software design methods are less robust than hardware, and software is becoming a dominating cause of systemwide failures. The software error rate is rising; people should not trust their results. Verification and validation are critical aspects for systems and applications codes. Many research software codes last upwards of 20-40 years across many teams of graduate student developers, while computing systems last 5 years. Developers should avoid embedding hardware aspects to improve portability of the codes across systems. Developers must also document and instrument their codes carefully for the next team to come along.

Checkpoint/restart: Most people are using checkpoint/restart, and some approaches are more efficient than others. But, global checkpoint/restart does not scale well to the emerging 100K processor and larger systems. It would be good to document the approaches and share them with the community to assist people in selecting the approach that may work best for their environment. The frequency at which checkpoints are done is based on the perceived failure rate of system and length of job, which leads into the topic in the next section about measurements.

Multicore capabilities: Many applications people are exploiting the multicore capabilities of the systems. Multithreaded cores and performance provide little performance benefit beyond two threads per core for many applications. As a result, some applications are using one core to monitor the system aspects for diagnosis, error checking, compiler reliability, performing compressions, etc.

What can systems do to better address fault tolerance?

Communications: Communications is a first step in the process. More extensive communications among the vendors, systems people, and applications teams is needed to understand the issues that causes faults and how best to address them on the different systems. The community, as a whole, needs to better understand the patterns of system use to improve fault prediction. More feedback is needed, between systems and applications and between systems and people. We need to recognize that it's *our* collective problem and that it spans both hardware and software.

Measurements: Measurements are a critical aspect to understanding fault tolerance, and not enough effort is being placed in how and what to measure. One cannot optimize for fault tolerance if one does not conduct measurements. Good measurements can help to identify the issues. Measurements should include incidents, occurrences, node-hours, etc. Some applications groups run validation suites such as conducting duplicate runs of the code with same input to compare results to detect issues. Some applications teams normally run ensembles (such as for weather forecasts) to conduct their research, which make it possible to compare the results of the runs to detect possible faults. While ensembles may be able to tolerate errors, it's important to understand the error bars for ensemble runs.

Preventative maintenance: Testing and preventive maintenance will help to reduce faults. Sites should conduct rigorous acceptance tests, conduct regular planned maintenance, and perform system verification tests after each maintenance period. Very little is understood about how to conduct preventative maintenance, especially for problems related to soft errors.

What are the challenges for the community?

Software development: Middleware developers should make develop specific techniques to write checkpoints at the speed of disk, though they may involve compromises that won't work for other types of I/O. Vertical integration is needed across hardware, software, message passing libraries, and numerical algorithms, to name a few. Coordinated Infrastructure for Fault Tolerance Systems (CIFTS) is a good start for interfaces for communicating with system components. A general model of failure demands a general model of recovery. There is a need for lightweight fault-tolerance interfaces to recover from errors with support for scalable archives for exascale and beyond. Fault-tolerant programming abstractions are needed to support applications developers. Resource-aware job scheduling is needed, as well as improved analysis and prediction tools.

Education: Many applications teams are unlikely to modify their code without clear benefit for getting more science done, or without assurance of the portability of the code to other systems. Public examples of successes are needed.

Fault detection & prediction: System complexity is leading to diminishing MTBF for large systems, which in some cases is less than 24 hours now, while on these same systems, job length is often 12-48 hours. Latency between fault and detection is a huge issue. The shorter the latency period, the better the recovery can be. The occurrence of silent corruption and/or software errors is difficult and slow to detect. It was suggested that we might expect one wrong answer per 10e6 processor hours. There is a consistent lack of standardized metrics, as well as an unwillingness of sites to share data. Sites seem to not want to "show their hand" in terms of the number of faults they are experiencing. The community needs to get this kind of data into the hands of researchers to help us better understand the nature and extent of the problem. The engineering cycle starts with recognizing one has a problem, identifying the problem, and then fixing it. In general, better reporting mechanisms are needed among all members of the community.

Emerging tools and techniques for managing faults: A variety of programming frameworks are emerging that can make it easier for an application to handle faults.

Charm++ uses a disk-based checkpoint, in-memory double checkpoint/restart, proactive object migration, redundant checkpoint on remote location on buddy processor, and a dummy process to replace a crashed processor. Global Arrays provide a virtual-machine-based checkpoint/restart, spare nodes to replace failed ones, and user-directed selection of global arrays to checkpoint. Implica uses fault-tolerant algorithms. Cactus provides checkpoint recovery in the framework between the system and the applications and knows more about the simulation than the system. These and other new programming paradigms will aid applications developers.

Good practices: Applications teams need to learn best-practices for application fault tolerance. Techniques to acquire include good fault-reporting practices, how to conduct checkpoint compression and optimization, and knowledge about fault-tolerance algorithm research. Application developers should collaborate with systems people to address fault tolerance. Applications must use checkpointing. Application developers should check error codes returned by libraries, rather than assuming that all routines will return correctly. Libraries exist that can assist with the issues, and tools to help are emerging. If nodes are writing local, asynchronous checkpoints, then it can be difficult and time consuming to reconstruct a consistent application state following a failure at scale (i.e., > 100k processors).

Next Steps

A series of meetings and workshops on fault tolerance and resiliency is currently occurring. These meetings are not formally coordinated, but there is a small set of organizers who talk informally about these workshops. The next meetings were as part of HPDC in Munich in June (Workshop on Resiliency in High-Performance Computing - http://xcr.cenit.latech.edu/resilience2009/) and a workshop in August in Washington, DC (National High Performance Computing Workshop on Resilience - http://institutes.lanl.gov/resilience/conferences/2009/).

A postworkshop poll of attendees found interest in the following ideas:

- Providing a large-scale test system, with capabilities for fault-injection
- Allowing users to try out the following on the test system:
    - Adam Moody's Scalable Checkpoint and Restart library on the test system.
    - Other in-memory checkpoint/recovery systems
    - The Charm++ work from Celso Mendes and U. Ill.
    - Products from the CIFTS project
    - Ideas related to residue codes for error detection and recovery
- Testing the prediction part of predictive failure analysis on a variety of systems – the recovery methods can be separated from failure prediction
- Creating a fault tolerance benchmark
    - Presumably, resource providers know what kind of errors are most likely to occur (node crash, memory error, interconnect failure, disk failure, etc.), and the benchmark could weigh these potential errors according to measured or predicted rates. (Note that these errors could be the result of either hardware or software faults.)
    - The benchmark would introduce certain kinds of faults (those that are deemed to be "important" or "realistic") into the system, at an "appropriate" level. (That level could be hardware, operating system, or system libraries such as MPI or libc.) One would then measure how the application behaves, whether it detects the error, whether it manages to recover, and how long this takes.
    - By standardizing on a certain set of faults, different fault-tolerance mechanisms could be compared,;and by providing a test harness (e.g., in the form of a software library), different applications could be tested comparatively easily.
    - Defining such a standard set will require some discussion, probably involving people who know what kinds of failures are common today (or will be common in a few years). Providing a test harness, especially if this can be implemented in software, would make it possible to test software before a new system is deployed.
- Taking stock of the efforts going on in the fields of resilience, fault tolerance, and reliability, and determining which aspects are in need of more emphasis and effort.
    - Government agencies need to step up and explain which aspects of these fields are most important to their missions.

After their respective talks, the BLCR (Paul Hargrove) and SCR (Adam Moody) speakers met and discussed briefly the possibility of using their respective software together. The outcome was a shared initial opinion that “it is just a matter of programming.” Both parties intend to follow up.

The next activity in this workshop series will likely be a workshop on parallel I/O (in 2010) that is now being planned. Interested people can contact us to be put on a mailing list for future workshops, announcements, etc. by sending e-mail to help@teragrid.org, referencing the TeraGrid eXtreme Scale Working Group (XSWG).

Conclusions

There is a large amount of interest in fault tolerance, resilience, and reliability of HPC systems in general. This interest is found at all levels of HPC, including systems, middleware, and applications. The only way to recover from faults is through the use of some redundancy, either in space or in time. Redundancy in time, in the form of writing checkpoints to disk and restarting at the most recent checkpoint after a fault that cause an application to crash/halt, is the most common tool used in applications today, but there are questions about how long this can continue to be a good solution as systems and memories grow faster than I/O bandwidth to disk. There is interest in both modifications to this, such as checkpoints to memory, partial checkpoints, and message logging, and alternative ideas, such as in-memory recovery using residues.

We believe that systematic exploration of these ideas holds the most promise for the scientific applications community. This would ideally involve the following steps:

1. Active participation by the petascale community in influencing the MPI-3 standards development now under way.
2. Collection of fault data from existing systems and projections from vendors of coming systems, in a common area, such as is being done now by the Computer Failure Data Repository (CFDR, http://cfdr.usenix.org/). This collection would benefit from requirements in contracts for new systems to collect such data and contribute it here.
3. Definition of a set of standard benchmarks for faults, including numbers, types, and rates, that should be expected.
4. Development of a test system, ideally with the following attributes:
    a. A variety of middleware systems that could be used for fault tolerance
    b. Sufficient capacity to run real applications at scale, either physically or virtually with runtimes within an order of magnitude of physical run times
    c. Capabilities for injecting numbers, types, and rates of faults to match the fault tolerance benchmarks
5. Exposure of this system to users, possibly through the TeraGrid, or some other national mechanism.
6. Discussion through on-line and in-person mechanisms of experiences with the test system.
7. Further planning after this initial exploration.

Fault tolerance has been an issue of discussion in the HPC community for at least the past 10 years, but much like other issues, the community has managed to put off addressing it during this period. As systems continue to grow to petascale and beyond, the field is approaching the point where we don't have any choice but to address this through R&D efforts.

Appendix A. Registered Attendees

Jay Alameda (NCSA) <jalameda@ncsa.uiuc.edu>
Dorian Arnold (UNM) <darnold@cs.unm.edu>
Galen Arnold (NCSA) <arnoldg@ncsa.uiuc.edu>
Sarala Arunagiri (UTEP) <sarunagiri@utep.edu>
Andrey Asadchev (Iowa) <asadchev@gmail.com>
David Bernholdt (ORNL) <bernholdtde@ornl.gov>
George Bosilca (UTK) <bosilca@eecs.utk.edu>
Patrick Bridges (UNM) <bridges@cs.unm.edu>
Chris Carothers (RPI) <chrisc@cs.rpi.edu>
Jonathan Cook (NMSU) <joncook@nmsu.edu>
John Daly (Center for Exceptional Computing) <john.t.daly@ugov.gov>
Nathan DeBardeleben (LANL) <ndebard@lanl.gov>
Kiran Desai (Iowa State) <kiran@cs.iastate.edu>
Tony Drummond (LBNL) <LADrummond@lbl.gov>
Peng Du (UTK) <du@cs.utk.edu>
Mootaz Elnozahy (IBM) <mootaz@us.ibm.com>
Christian Engelmann (ORNL) <engelmannc@ornl.gov>
Robert Fiedler (NCSA) <rfiedler@uiuc.edu>
Garth Gibson (CMU) <garth.gibson@cs.cmu.edu>
Rinku Gupta (ANL) <rgupta@mcs.anl.gov>
Thomas Hacker (Purdue) <tjhacker@purdue.edu>
Paul Hargrove (LBNL) <phhargrove@lbl.gov>
Michael Heroux (SNL) <maherou@sandia.gov>
Patty Hough (Sandia) <pdhough@ca.sandia.gov>
Zbigniew Kalbarczyk (U. Illinois) <kalbar@crhc.illinois.edu>
Larry Kaplan (Cray) <lkaplan@cray.com>
Daniel S. Katz (CCT) <dsk@cct.lsu.edu>
Bill Kramer (NCSA) <wkramer@ncsa.uiuc.edu>
Kenneth Koch (LANL) <krk@lanl.gov>
Patricia Kovatch (NICS) <pkovatch@utk.edu>
Sriram Krishnamoorthy (PNL) <sriram.krishnamoorthy@pnl.gov>
Don Lamb (U. of Chicago) <d-lamb@uchicago.edu>
Scott Lathrop (NCSA) <scott@ncsa.uiuc.edu>
Elizabeth Leake (Chicago) <eleake@uchicago.edu>
Phil Maechling (USC) <maechlin@usc.edu>
Amitava Majumdar (SCSD) <majumdar@sdsc.edu>
Celso Mendes (UIUC) <cmendes@illinois.edu>
Kent Milfeld (TACC) <milfeld@tacc.utexas.edu>
Adam Moody (LLNL) <moody20@llnl.gov>
Nick Nystom (PSC) <nystrom@psc.edu>
Ron Oldfield (SNL) < raoldfi@sandia.gov>
Craig Rasmussen (LANL) <crasmussen@newmexicoconsortium.org>
Joachim (Jimmy) Raeder (U. New Hampshire) <J.Raeder@unh.edu>
Erik Schnetter (LSU) <schnetter@cct.lsu.edu>
Eric Schrock (Sun) <eschrock@sun.com>
Stephen Scott (ORNL) <scottsl@ornl.gov>
Jon Stearley (Sandia) <jrstear@sandia.gov>
Narate Taerat (LaTech) <nta008@latech.edu>
Sudharshan Vazhkudai (ORNL) <vazhkudaiss@ornl.gov>
Lawrence Votta (CREATE) <votta@alum.mit.edu>
Patrick Widener (LANL) <pmw@cs.unm.edu>
Matthew Woitaszek (UCAR) <matthew@ucar.edu>

Appendix B. Workshop Agenda

Thursday, March 19

8:00 - Opening/Context - BW/TG short introductions

8:30 - Fault Tolerance 101, Zbigniew Kalbarczyk (U. Illinois)

9:00 - Large Systems Session I
9:00 - Faults and Fault-Tolerance on the Argonne BG/P System, Rinku Gupta (ANL)
9:25 - Essential Feedback Loops, Jon Stearley (SNL)
Questions for each speaker:
- What can you tell us about your fault/error situation currently?
- What types of errors do you see?
- What rate of errors?
- What are you worried about in the future?
- What do you think is needed to help?
- What do you think is reasonable for apps people to do?

9:50 - break

10:20 - Large Systems Session II
10:20 - Experiences with Kraken, Patricia Kovatch (NICS)
10:55 - Growing Pains of Petascale Computing: Integrating Hardware, Software and Middleware for Successful Capacity and Throughput, Kent Milfeld (TACC)
11:10 - Rollback-Recovery in the Petascale Era, Mootaz Elnozahy (IBM)
11:35 - Finding and Preventing Faults on Extreme Scale Systems, Bill Kramer (NCSA)
12:00 - discussion
Questions for each speaker:
- What can you tell us about your fault/error situation currently?
- What types of errors do you see?
- What rate of errors?
- What are you worried about in the future?
- What do you think is needed to help?
- What do you think is reasonable for apps people to do?

12:30 - lunch

1:30 - I/O system session
1:30 - The Role of Storage in Exascale Fault Tolerance, Garth Gibson (CMU)
1:55 - stdchk: A Checkpoint Storage System for HPC Applications, Sudharshan Vazhkudai (ORNL)
2:10 - I/O Fault Diagnosis in Software Storage Systems, Eric Schrock (Sun)
2:45 - discussion
Questions for each speaker:
- What can you tell us about your fault/error situation currently?
- What types of errors do you see?
- What rate of errors?
- What are you worried about in the future?
- What do you think is needed to help?
- What do you think is reasonable for apps people to do?

3:00 - break

3:30 - FT tech session
3:30 - System-Level Checkpoint/Restart with BLCR, Paul Hargrove (LBL)
3:55 - Scalable Fault Tolerance Schemes Using Adaptive Runtime Support, Celso Mendes (U. Illinois)
4:20 - Handling Faults in a Global Address Space Programming Model, Sriram Krishnamoorthy (PNL)
4:45 - Implications of System Errors in the Context of Numerical Accuracy, Patty Hough (SNL)
5:10 - The Scalable Checkpoint / Restart (SCR) Library: Approaching File I/O Bandwidth of 1 TB/s, Adam Moody (LLNL)
5:35 - discussion
Questions for each speaker
- What have you done that people should be aware of?
- How have you tested it?
- Who's using it?
- What do you want from users to help you develop/test it?
- What do you think is reasonable for apps people to do?
- What errors/faults do you want to be aware of/notified of?

6:30 - dinner

Friday, March 20
8:30 - MIxed FT tech / app session
8:30 - Sustained Exascale: The Challenge, George Bosilca (U. Tennessee)
8:55 - Between Application and System: Fault Tolerance Mechanisms for the Cactus Software Framework, Erik Schnetter (LSU)
9:20 - Fault Tolerance Support for HPC Tools and Applications: Scalability and Sustainability, Tony Drummond (LBL)
9:45 - Fault Tolerance in CREATE Phase 1, Lawrence Votta
10:10 - discussion
Questions for each FT speaker
What have you done that people should be aware of?
How have you tested it?
Who's using it?
What do you want from users to help you develop/test it?
What do you think is reasonable for apps people to do?
What errors/faults do you want to be aware of/notified of?
Questions for each app speaker:
What science/computing are you doing now, focused on computing more than science?
What are you worried about, particularly thinking of future grand challenge science?
What errors/faults do you want to be aware of/notified of?
What do you want from tools/technologies?
What have you done about it so far?
What are you planning to do?
What do you think is reasonable for apps people to do?

10:30 - break

11:00 - apps session
11:00 - Application Resilience for Truculent Systems, John Daly (Center for Exceptional Computing)
11:25 - Insights into Fault Tolerance from FLASH Production Runs on Top 3 Supercomputing Platforms, Don Lamb (U. Chicago)
11:50 - An Analysis of Clustered Failures on Blue Gene Supercomputing Systems, Christopher Carothers (RPI) & Thomas Hacker (Purdue)
12:15 - discussion
Questions for each app speaker:
What science/computing are you doing now, focused on computing more than science?
What are you worried about, particularly thinking of future grand challenge science?
What errors/faults do you want to be aware of/notified of?
What do you want from tools/technologies?
What have you done about it so far?
What are you planning to do?
What do you think is reasonable for apps people to do?
12:30 - closing discussion

1:00 - end

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439-4847

www.anl.gov